Preliminary research on the implementation method of Ethereum compatible chain

Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 2024-3-25 | 0.1 | Initial Draft | Xun Wang |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[Table of Contents 2](#_Toc2688)

[1. Objectives 3](#_Toc5315)

[1.1 References 3](#_Toc3332)

# Objectives

1. Figure out how to implement an Ethereum-compatible chain.

2. Write documentation on how to implement an Ethereum-compatible chain.

3. Extract and package Besu and RPC-related code into a standalone service. Compile and run this service, and validate its functionality by connecting and testing it through the terminal.

4 You can use ChatGPT's external brain to help solve problems, implement solutions, and code.

If you need assistance with problem-solving, solution implementation, or coding(Explain Besu's RPC code, provide demand solutions and implementation code according to needs, etc.), feel free to ask ChatGPT first!

5 Use Maven to manage source code.

## References

[Generative AI in the Development and Operations - DZone](https://dzone.com/articles/potential-benefits-of-using-generative-ai-in-the-d)

[前 20 个 EVM 兼容区块链 (plisio.net)](https://plisio.net/zh/blog/the-top-20-evm-compatible-blockchains)

[支持的RPC方法 | Moonbeam Docs](https://docs.moonbeam.network/cn/builders/get-started/eth-compare/rpc-support/)

[ChainList](https://chainlist.org/)

[JSON-RPC API | ethereum.org](https://ethereum.org/en/developers/docs/apis/json-rpc/)

[Welcome | Besu documentation (hyperledger.org)](https://besu.hyperledger.org/)

[ethereum.github.io/execution-apis/api-documentation/](https://ethereum.github.io/execution-apis/api-documentation/)

[besu/ethereum/api/src/main/java/org/hyperledger/besu/ethereum/api/jsonrpc/RpcApis.java at bb9ba13cf8fec3a8f2bc15d1fd7cc2cdd6c12da7 · hyperledger/besu (github.com)](https://github.com/hyperledger/besu/blob/bb9ba13cf8fec3a8f2bc15d1fd7cc2cdd6c12da7/ethereum/api/src/main/java/org/hyperledger/besu/ethereum/api/jsonrpc/RpcApis.java" \l "L43)

[besu/ethereum/api/src/main/java/org/hyperledger/besu/ethereum/api/jsonrpc at bb9ba13cf8fec3a8f2bc15d1fd7cc2cdd6c12da7 · hyperledger/besu (github.com)](https://github.com/hyperledger/besu/tree/bb9ba13cf8fec3a8f2bc15d1fd7cc2cdd6c12da7/ethereum/api/src/main/java/org/hyperledger/besu/ethereum/api/jsonrpc)

Certainly, here's the archive of your three questions along with my responses in English:

---

\*\*Question 1: Can the RPC module of Besu be run independently?\*\*

Yes, you can run the RPC module of Besu independently. Besu is an Ethereum client that can be run as a standalone node, and its RPC (Remote Procedure Call) module allows you to communicate with the node over a network interface, performing various operations such as querying blockchain data, sending transactions, etc.

To run the RPC module of Besu independently, you need to follow these steps:

1. \*\*Install Besu\*\*: Firstly, you need to download and install the Besu client. You can obtain the latest version of the Besu client from the official website or GitHub repository and follow the instructions for installation.

2. \*\*Configure Besu\*\*: After installation, you need to configure Besu to enable the RPC module. You can edit the Besu configuration file to specify the RPC port, allowed request origins, etc. You can also configure Besu through command-line options, depending on your preferences and requirements.

3. \*\*Start Besu\*\*: Once configured, you can start the Besu client. When starting, ensure that you have specified the correct configuration file or command-line options to ensure that the RPC module is properly enabled.

4. \*\*Test RPC Connection\*\*: Once Besu is up and running, you can use any client or tool that supports RPC to connect and communicate with it. For example, you can use cURL, Postman, Web3.js, etc., to send RPC requests, query blockchain data, or execute transactions.

Please note to consider security when configuring Besu and take appropriate measures to protect your node and RPC interface from unauthorized access or attacks.

---

\*\*Question 2: What is the package for Besu's RPC module?\*\*

Besu's RPC module is implemented through plugins, and its related code can be found in the `besu-plugins` repository. Specifically, the code for the RPC module is located within the `besu-plugin-rpc` submodule. In this submodule, you can find Java classes that implement RPC functionality and other related code.

You can find the code for this submodule in Besu's GitHub repository: [https://github.com/hyperledger/besu/tree/main/plugins/rpc](https://github.com/hyperledger/besu/tree/main/plugins/rpc)

---

\*\*Question 3: Design principles and architecture of Besu's plugin system\*\*

The design principles and architecture of Besu's plugin system are crucial aspects that enable developers to extend and customize Besu's functionality through plugins. Here's an overview of the design principles and architecture:

\*\*Architecture Design:\*\*

1. \*\*Modular Design\*\*: Besu adopts a modular design that allows various functional units to be extended and customized through plugins. This keeps Besu's core code concise, flexible, while providing good extensibility and customizability.

2. \*\*Plugin Interfaces\*\*: Besu defines a series of plugin interfaces for different types of plugins, such as RPC, synchronization, consensus, etc. These interfaces specify the methods and functionalities that plugins need to implement, providing developers with a unified development standard.

3. \*\*Dependency Injection\*\*: Besu uses dependency injection to manage dependencies between plugins. This allows plugins to collaborate, share resources, and dynamically add or remove plugins at runtime.

4. \*\*Scalability\*\*: Besu's plugin architecture design has good scalability, allowing developers to add new plugins or customize existing ones according to their needs without modifying Besu's core code.

\*\*Plugin System Principles:\*\*

1. \*\*Plugin Loading\*\*: Besu scans a specified directory during startup to load all plugins that comply with the specification. Each plugin has a descriptor file containing metadata information and dependencies.

2. \*\*Plugin Lifecycle Management\*\*: Besu manages the lifecycle of plugins, including initialization, startup, shutdown, etc. Plugins can perform initialization or cleanup tasks during startup and shutdown.

3. \*\*Plugin Communication\*\*: Besu provides a mechanism for plugins to communicate with each other, enabling collaboration. Typically, plugins obtain instances of other plugins through dependency injection and call their methods to achieve communication and collaboration.

4. \*\*Security\*\*: Besu's plugin architecture design considers security factors, ensuring that plugin loading and execution do not pose threats to the system's security. Plugin permissions and access control are strictly managed and restricted.

In summary, Besu's plugin architecture design and principles provide a flexible, extensible way for developers to customize and extend Besu's functionality while maintaining system stability and security.

---

This should provide a comprehensive English version of your questions and my responses. If you need further clarification or have any more questions, feel free to ask!

Certainly! Here's the summary of the question and answer in English:

---

\*\*Question:\*\* What technology framework is used to implement Besu's RPC?

\*\*Answer:\*\* Besu's RPC module is implemented using Java language and the Spring framework. Spring is an open-source Java application framework that provides a wide range of functionalities, including dependency injection, aspect-oriented programming, data access, transaction management, etc. In Besu, the Spring framework is utilized to implement the RPC module's functionalities, such as defining RPC interfaces, handling RPC requests, managing dependencies, etc. The flexibility and powerful features of the Spring framework enable Besu to develop and maintain the RPC module quickly, with good scalability and performance.

---

If you need further clarification or have any more questions, feel free to ask!

Here's the compilation of the question and answer in English:

---

\*\*Question:\*\* How to extract and create a standalone project from Besu's RPC-related source code?

\*\*Answer:\*\* To extract Besu's RPC-related source code and create a standalone project, you can follow these steps:

1. \*\*Identify RPC Module Code Location\*\*: First, locate the RPC module's code within the Besu code repository. Typically, the RPC module code resides in a specific directory within the source code tree.

2. \*\*Copy Relevant Code\*\*: Find the RPC module's code in the Besu code repository and copy it to a new directory, serving as the starting point for your standalone project.

3. \*\*Create a New Standalone Project\*\*: Within the chosen directory, create a new standalone project. You can use Maven, Gradle, or another build tool to manage your project.

4. \*\*Adjust Directory Structure\*\*: Adjust the directory structure of the RPC module code according to your project structure and the requirements of your build tool. Ensure that all source code, resource files, and configuration files are in their correct locations.

5. \*\*Update Dependencies and Configuration\*\*: Update dependencies and configurations of the RPC module to suit the new project's requirements. This may involve modifying dependency versions, adjusting file paths in configuration files, etc.

6. \*\*Build and Test\*\*: Build the new standalone project using your chosen build tool. Ensure that everything compiles and builds successfully. Then, run tests to verify that the RPC module's functionality is intact.

7. \*\*Run the Standalone Project\*\*: Once your standalone project builds successfully and passes tests, you can run it. Depending on your needs, you may need to write startup scripts or configuration files to set up the runtime environment for the RPC module.

By following these steps, you should be able to extract Besu's RPC-related source code and create a standalone project that can be run independently. This allows for more flexibility in using and customizing the RPC module without relying on the entire Besu codebase.

---

If you need further clarification or have any more questions, feel free to ask!

Question 1: "Where is the RPC plugin of Besu started from in terms of source code?"

Answer 1: "In Besu's source code, the RPC plugin is typically started and managed by implementing the JsonRpcHttpService interface. You can find the part responsible for starting the RPC plugin, usually located in the JsonRpcHttpService class or its implementation classes."

Here is a simple example demonstrating how to start a `JsonRpcHttpService` from the `main` method:

```java

import org.hyperledger.besu.ethereum.jsonrpc.JsonRpcConfiguration;

import org.hyperledger.besu.ethereum.jsonrpc.JsonRpcHttpService;

import org.hyperledger.besu.ethereum.jsonrpc.JsonRpcHttpServiceFactory;

import org.hyperledger.besu.ethereum.jsonrpc.RpcApis;

public class Main {

public static void main(String[] args) {

// Create a JsonRpcConfiguration object to configure the RPC service parameters

JsonRpcConfiguration rpcConfiguration = JsonRpcConfiguration.createDefault();

// Add the RPC APIs to be enabled

rpcConfiguration.setRpcApis(RpcApis.DEFAULT\_JSON\_RPC\_APIS);

// Set the port for the RPC service

rpcConfiguration.setPort(8545);

// Create a JsonRpcHttpServiceFactory instance

JsonRpcHttpServiceFactory rpcHttpServiceFactory = new JsonRpcHttpServiceFactory();

// Create a JsonRpcHttpService instance based on the configuration

JsonRpcHttpService rpcHttpService = rpcHttpServiceFactory.create(rpcConfiguration);

// Start the RPC service

rpcHttpService.start().join();

// Output a message indicating that the JSON-RPC service has started and print the port on which the service is running

System.out.println("JSON-RPC service started on port " + rpcHttpService.getPort());

// Wait for the RPC service to stop

rpcHttpService.awaitStop();

}

}

```

In this example:

- A `JsonRpcConfiguration` object is created to configure the parameters of the RPC service, such as the RPC APIs to enable and the port number.

- A `JsonRpcHttpServiceFactory` instance is created to create a `JsonRpcHttpService` instance.

- A `JsonRpcHttpService` instance is created based on the configuration parameters.

- The RPC service is started using the `start()` method, and the service start completion is awaited.

- A message is output indicating that the JSON-RPC service has started, and the port on which the service is running is printed.

- The program waits for the RPC service to stop using the `awaitStop()` method.

You can adjust the configuration parameters as needed and embed this code snippet into your project to start the `JsonRpcHttpService` from the `main` method.